autogluon_each_location

October 21, 2023

1 Config

[1]:
```
# config

label = 'y'
metric = 'mean_absolute_error'
time_limit = 60*5
presets = 'best_quality'

do_drop_ds = True
# hour, dayofweek, dayofmonth, month, year
use_dt_attrs = []#["hour", "year"]
use_estimated_diff_attr = False
use_is_estimated_attr = True

to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:ms",␣
↪"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:cm",␣
↪"wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:mm",␣
↪"rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm", "absolute_humidity_2m:
↪gm3", "air_density_2m:kgm3"]#, "msl_pressure:hPa", "pressure_50m:hPa", ␣
↪"pressure_100m:hPa"]

#to_drop = ["snow_drift:idx", "snow_density:kgm3", "wind_speed_w_1000hPa:
↪ms",␣"dew_or_rime:idx", "prob_rime:p", "fresh_snow_12h:cm", "fresh_snow_24h:
↪cm",␣"wind_speed_u_10m:ms", "wind_speed_v_10m:ms", "snow_melt_10min:
↪mm",␣"rain_water:kgm2", "dew_point_2m:K", "precip_5min:mm",␣
↪"absolute_humidity_2m:gm3", "air_density_2m:kgm3"]

use_groups = False
n_groups = 8

auto_stack = False
num_stack_levels = 0
num_bag_folds = 8
num_bag_sets = 20

use_tune_data = True
```

```
use_test_data = True
tune_and_test_length = 0.5 # 3 months from end
holdout_frac = None
use_bag_holdout = True # Enable this if there is a large gap between score_val␣
↪and score_test in stack models.

sample_weight = None#'sample_weight' #None
weight_evaluation = False#
sample_weight_estimated = 1
sample_weight_may_july = 1

run_analysis = False

shift_predictions_by_average_of_negatives_then_clip = False
clip_predictions = True
shift_predictions = False
```

2 Loading and preprocessing

[2]:
```
import pandas as pd
import numpy as np




import warnings
warnings.filterwarnings("ignore")



def feature_engineering(X):
    # shift all columns with "1h" in them by 1 hour, so that for index 16:00,␣
↪we have the values from 17:00
    # but only for the columns with "1h" in the name
    #X_shifted = X.filter(regex="\dh").shift(-1, axis=1)
    #print(f"Number of columns with 1h in name: {X_shifted.columns}")


    columns = ['clear_sky_energy_1h:J', 'diffuse_rad_1h:J', 'direct_rad_1h:J',
               'fresh_snow_12h:cm', 'fresh_snow_1h:cm', 'fresh_snow_24h:cm',
               'fresh_snow_3h:cm', 'fresh_snow_6h:cm']

    # Filter rows where index.minute == 0
    X_shifted = X[X.index.minute == 0][columns].copy()

    # Create a set for constant-time lookup
    index_set = set(X.index)
```

```
    # Vectorized time shifting
    one_hour = pd.Timedelta('1 hour')
    shifted_indices = X_shifted.index + one_hour
    X_shifted.loc[shifted_indices.isin(index_set)] = X.
 ↪loc[shifted_indices[shifted_indices.isin(index_set)]]

    # Count
    count1 = len(shifted_indices[shifted_indices.isin(index_set)])
    count2 = len(X_shifted) - count1

    print("COUNT1", count1)
    print("COUNT2", count2)

    # Rename columns
    X_old_unshifted = X_shifted.copy()
    X_old_unshifted.columns = [f"{col}_not_shifted" for col in X_old_unshifted.
 ↪columns]

    # Resampling
    X = X.resample('H').mean()

    # Update columns
    X[columns] = X_shifted

    # If 'date_calc' is present, handle it
    if 'date_calc' in X.columns:
        date_calc = X[X.index.minute == 0]['date_calc']
        X['date_calc'] = date_calc

    # resample to hourly
    print("index: ", X.index[0])
    X = X.resample('H').mean()
    print("index AFTER: ", X.index[0])

    X[columns] = X_shifted[columns]
    #X[X_old_unshifted.columns] = X_old_unshifted

    if date_calc is not None:
        X['date_calc'] = date_calc

    return X




def fix_X(X, name):
```

```python
    # Convert 'date_forecast' to datetime format and replace original column␣
↪with 'ds'
    X['ds'] = pd.to_datetime(X['date_forecast'])
    X.drop(columns=['date_forecast'], inplace=True, errors='ignore')
    X.sort_values(by='ds', inplace=True)
    X.set_index('ds', inplace=True)


    X = feature_engineering(X)

    return X



def handle_features(X_train_observed, X_train_estimated, X_test, y_train):
    X_train_observed = fix_X(X_train_observed, "X_train_observed")
    X_train_estimated = fix_X(X_train_estimated, "X_train_estimated")
    X_test = fix_X(X_test, "X_test")


    if weight_evaluation:
        # add sample weights, which are 1 for observed and 3 for estimated
        X_train_observed["sample_weight"] = 1
        X_train_estimated["sample_weight"] = sample_weight_estimated
        X_test["sample_weight"] = sample_weight_estimated


    y_train['ds'] = pd.to_datetime(y_train['time'])
    y_train.drop(columns=['time'], inplace=True)
    y_train.sort_values(by='ds', inplace=True)
    y_train.set_index('ds', inplace=True)

    return X_train_observed, X_train_estimated, X_test, y_train




def preprocess_data(X_train_observed, X_train_estimated, X_test, y_train,␣
↪location):
    # convert to datetime
    X_train_observed, X_train_estimated, X_test, y_train =␣
↪handle_features(X_train_observed, X_train_estimated, X_test, y_train)

    if use_estimated_diff_attr:
        X_train_observed["estimated_diff_hours"] = 0
        X_train_estimated["estimated_diff_hours"] = (X_train_estimated.index -␣
↪pd.to_datetime(X_train_estimated["date_calc"])).dt.total_seconds() / 3600
```

```
        X_test["estimated_diff_hours"] = (X_test.index - pd.
↪to_datetime(X_test["date_calc"])).dt.total_seconds() / 3600

        X_train_estimated["estimated_diff_hours"] =␣
↪X_train_estimated["estimated_diff_hours"].astype('int64')
        # the filled once will get dropped later anyways, when we drop y nans
        X_test["estimated_diff_hours"] = X_test["estimated_diff_hours"].
↪fillna(-50).astype('int64')

    if use_is_estimated_attr:
        X_train_observed["is_estimated"] = 0
        X_train_estimated["is_estimated"] = 1
        X_test["is_estimated"] = 1

    # drop date_calc
    X_train_estimated.drop(columns=['date_calc'], inplace=True)
    X_test.drop(columns=['date_calc'], inplace=True)


    y_train["y"] = y_train["pv_measurement"].astype('float64')
    y_train.drop(columns=['pv_measurement'], inplace=True)
    X_train = pd.concat([X_train_observed, X_train_estimated])


    # clip all y values to 0 if negative
    y_train["y"] = y_train["y"].clip(lower=0)

    X_train = pd.merge(X_train, y_train, how="inner", left_index=True,␣
↪right_index=True)

    # print number of nans in y
    print(f"Number of nans in y: {X_train['y'].isna().sum()}")


    X_train["location"] = location
    X_test["location"] = location

    return X_train, X_test
# Define locations
locations = ['A', 'B', 'C']

X_trains = []
X_tests = []
# Loop through locations
for loc in locations:
    print(f"Processing location {loc}...")
    # Read target training data
```

```
    y_train = pd.read_parquet(f'{loc}/train_targets.parquet')

    # Read estimated training data and add location feature
    X_train_estimated = pd.read_parquet(f'{loc}/X_train_estimated.parquet')

    # Read observed training data and add location feature
    X_train_observed= pd.read_parquet(f'{loc}/X_train_observed.parquet')

    # Read estimated test data and add location feature
    X_test_estimated = pd.read_parquet(f'{loc}/X_test_estimated.parquet')

    # Preprocess data
    X_train, X_test = preprocess_data(X_train_observed, X_train_estimated,␣
↪X_test_estimated, y_train, loc)

    X_trains.append(X_train)
    X_tests.append(X_test)

# Concatenate all data and save to csv
X_train = pd.concat(X_trains)
X_test = pd.concat(X_tests)
```

```
Processing location A…
COUNT1 29667
COUNT2 1
index:  2019-06-02 22:00:00
index AFTER:  2019-06-02 22:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 0
Processing location B…
COUNT1 29232
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
```

```
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 4
Processing location C…
COUNT1 29206
COUNT2 1
index:  2019-01-01 00:00:00
index AFTER:  2019-01-01 00:00:00
COUNT1 4392
COUNT2 2
index:  2022-10-28 22:00:00
index AFTER:  2022-10-28 22:00:00
COUNT1 702
COUNT2 18
index:  2023-05-01 00:00:00
index AFTER:  2023-05-01 00:00:00
Number of nans in y: 6059
```

2.1 Feature enginering

2.1.1 Remove anomalies

[3]:
```
import numpy as np
import pandas as pd


# loop thorugh x train[y], keep track of streaks of same values and replace␣
↪them with nan if they are too long
# also replace nan with 0

import numpy as np

def replace_streaks_with_nan(df, max_streak_length, column="y"):
    for location in df["location"].unique():
        x = df[df["location"] == location][column].copy()

        last_val = None
        streak_length = 1
        streak_indices = []
        allowed = [0]
        found_streaks = {}

        for idx in x.index:
            value = x[idx]
            # if location == "B":
            #     continue

            if value == last_val and value not in allowed:
                streak_length += 1
```

```
                streak_indices.append(idx)
            else:
                streak_length = 1
                last_val = value
                streak_indices.clear()

            if streak_length > max_streak_length:
                found_streaks[value] = streak_length

                for streak_idx in streak_indices:
                    x[idx] = np.nan
                streak_indices.clear()  # clear after setting to NaN to avoid␣
↪setting multiple times
        df.loc[df["location"] == location, column] = x

        print(f"Found streaks for location {location}: {found_streaks}")

    return df


# deep copy of X_train into x_copy
X_train = replace_streaks_with_nan(X_train.copy(), 3, "y")
```

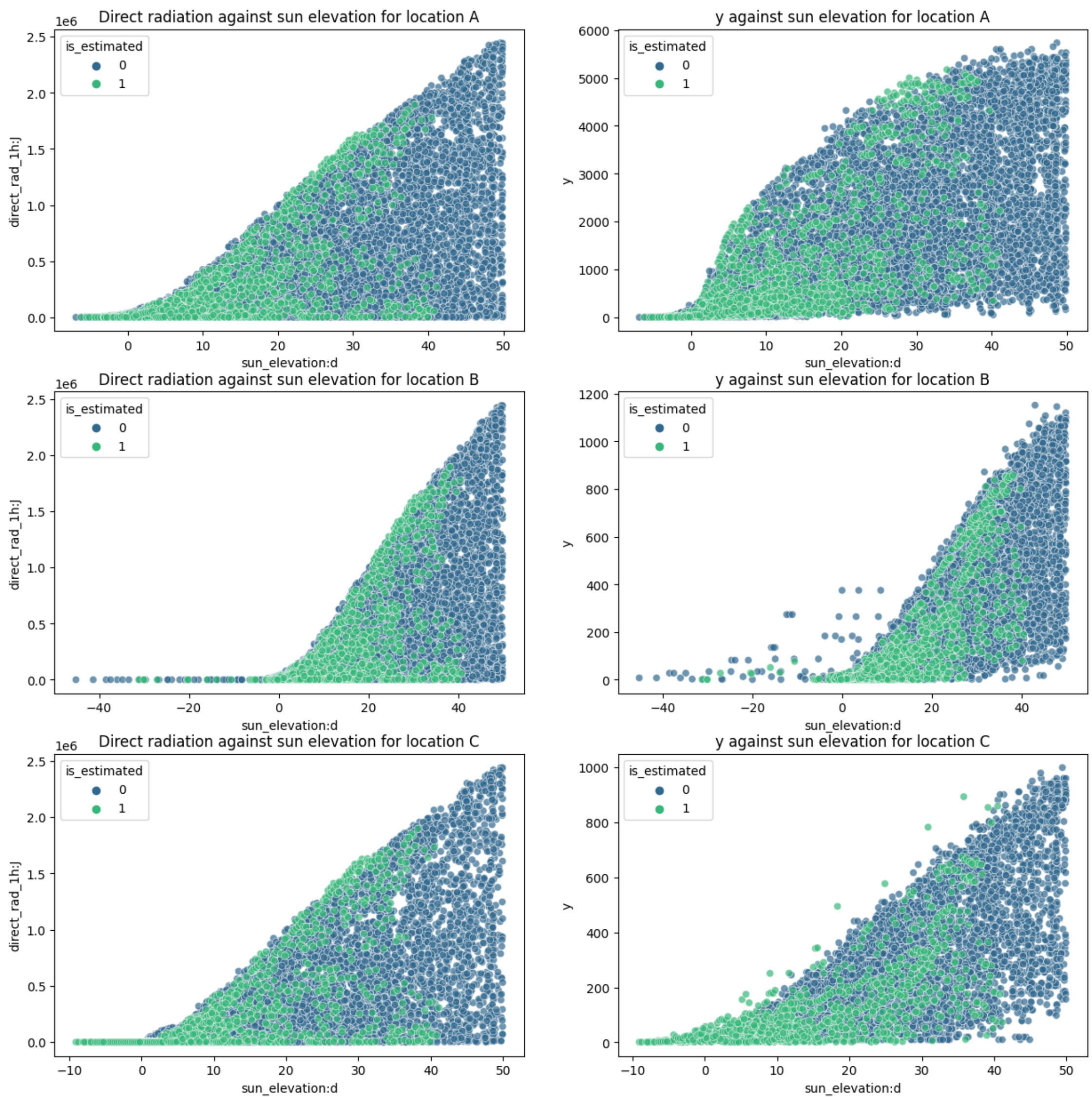
```
Found streaks for location A: {}
Found streaks for location B: {3.45: 28, 6.9: 7, 12.9375: 5, 13.8: 8, 276.0: 78,
18.975: 58, 0.8625: 4, 118.1625: 33, 34.5: 11, 183.7125: 1058, 87.1125: 7,
79.35: 34, 7.7625: 12, 27.6: 448, 273.41249999999997: 72, 264.78749999999997:
55, 169.05: 33, 375.1875: 56, 314.8125: 66, 76.7625: 10, 135.4125: 216, 81.9375:
202, 2.5875: 12, 81.075: 210}
Found streaks for location C: {9.8: 4, 29.400000000000002: 4, 19.6: 4}
```

[4]:
```
# print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

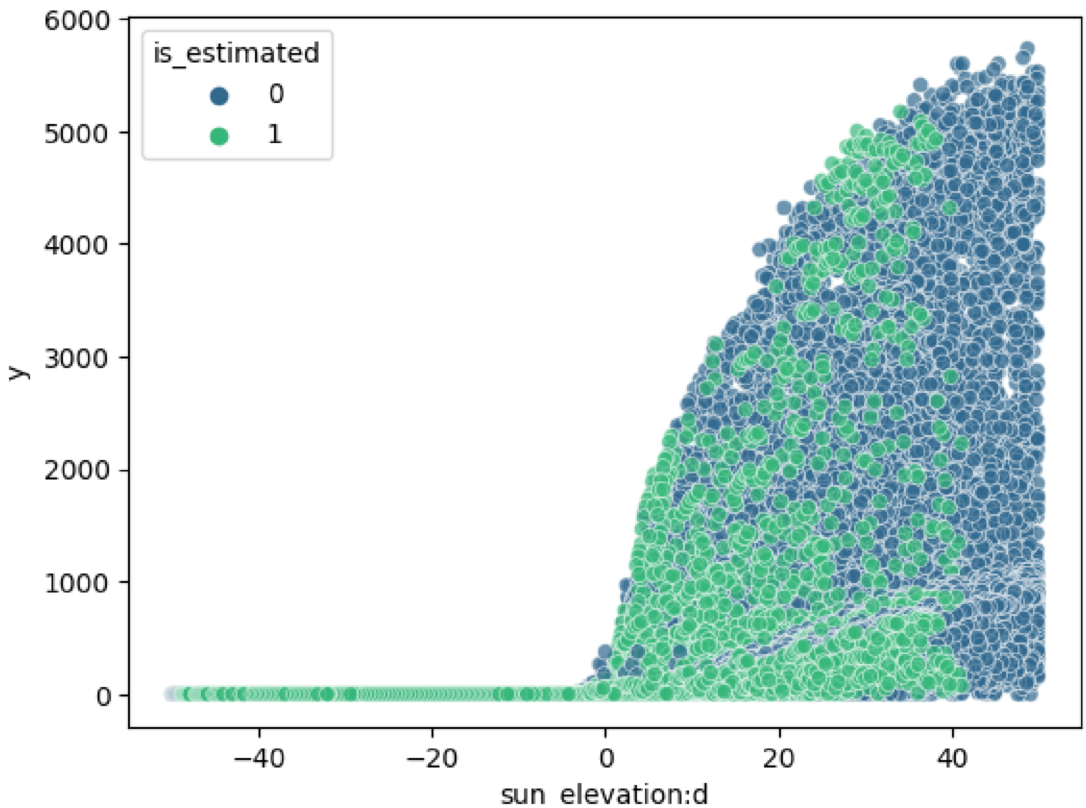
```
Dropped rows:  9291
```

[5]:
```
import matplotlib.pyplot as plt
import seaborn as sns
# Filter out rows where y == 0
temp = X_train[X_train["y"] != 0]

# Plotting
fig, axes = plt.subplots(len(locations), 2, figsize=(15, 5 * len(locations)))

for idx, location in enumerate(locations):
```

```python
    sns.scatterplot(ax=axes[idx][0], data=temp[temp["location"] == location], x="sun_elevation:d", y="direct_rad_1h:J", hue="is_estimated", palette="viridis", alpha=0.7)
    axes[idx][0].set_title(f"Direct radiation against sun elevation for location {location}")

    sns.scatterplot(ax=axes[idx][1], data=temp[temp["location"] == location], x="sun_elevation:d", y="y", hue="is_estimated", palette="viridis", alpha=0.7)
    axes[idx][1].set_title(f"y against sun elevation for location {location}")

# plt.tight_layout()
# plt.show()
```

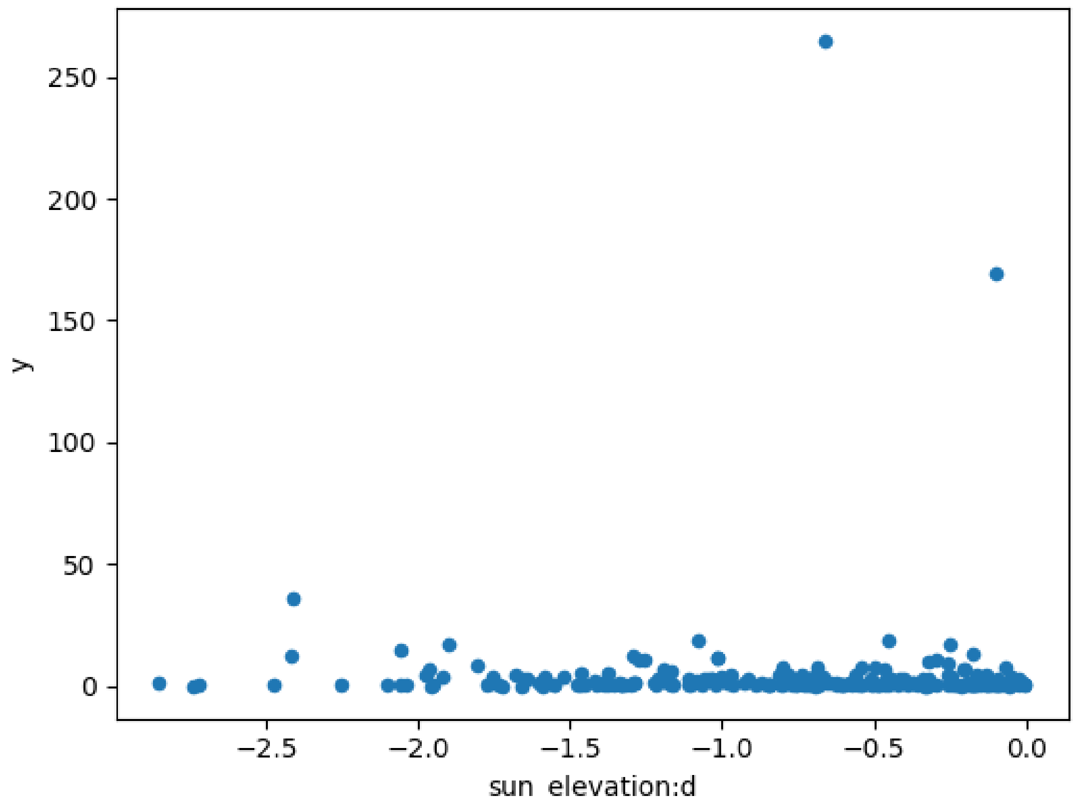


[6]:
```
thresh = 0.1

# Update "y" values to NaN if they don't meet the criteria
mask = (X_train["direct_rad_1h:J"] <= thresh) & (X_train["diffuse_rad_1h:J"] <=␣
↪thresh) & (X_train["y"] >= 0.1)
X_train.loc[mask, "y"] = np.nan

# Plot using sns scatterplot
sns.scatterplot(data=X_train, x="sun_elevation:d", y="y", hue="is_estimated",␣
↪palette="viridis", alpha=0.7)
plt.show()
```



[7]:
```
# location B count number of rows with y > 0 and sun_elevation:d < 0

condition = (X_train["location"] == "B") & (X_train["y"] > 0) &␣
↪(X_train["sun_elevation:d"] < 0)
bad = X_train[condition]

bad.plot.scatter(x="sun_elevation:d", y="y")
```

[7]: <AxesSubplot: xlabel='sun_elevation:d', ylabel='y'>



[8]:
```
# print num rows
temprows = len(X_train)
X_train.dropna(subset=['y', 'direct_rad_1h:J', 'diffuse_rad_1h:J'],␣
↪inplace=True)
print("Dropped rows: ", temprows - len(X_train))
```

```
Dropped rows:  356
```

2.1.2 Other stuff

[9]:
```
import numpy as np
import pandas as pd

for attr in use_dt_attrs:
    X_train[attr] = getattr(X_train.index, attr)
    X_test[attr] = getattr(X_test.index, attr)

#print(X_train.head())
```

```python
# If the "sample_weight" column is present and weight_evaluation is True, multiply sample_weight with sample_weight_may_july if the ds is between 05-01 00:00:00 and 07-03 23:00:00, else add sample_weight as a column to X_train
if weight_evaluation:
    if "sample_weight" not in X_train.columns:
        X_train["sample_weight"] = 1

    X_train.loc[((X_train.index.month >= 5) & (X_train.index.month <= 6)) | ((X_train.index.month == 7) & (X_train.index.day <= 3)), "sample_weight"] *= sample_weight_may_july


print(X_train.iloc[200])
print(X_train[((X_train.index.month >= 5) & (X_train.index.month <= 6)) | ((X_train.index.month == 7) & (X_train.index.day <= 3))].head(1))



if use_groups:
    # fix groups for cross validation
    locations = X_train['location'].unique()  # Assuming 'location' is the name of the column representing locations

    grouped_dfs = []  # To store data frames split by location

    # Loop through each unique location
    for loc in locations:
        loc_df = X_train[X_train['location'] == loc]

        # Sort the DataFrame for this location by the time column
        loc_df = loc_df.sort_index()

        # Calculate the size of each group for this location
        group_size = len(loc_df) // n_groups

        # Create a new 'group' column for this location
        loc_df['group'] = np.repeat(range(n_groups), repeats=[group_size]*(n_groups-1) + [len(loc_df) - group_size*(n_groups-1)])

        # Append to list of grouped DataFrames
        grouped_dfs.append(loc_df)

    # Concatenate all the grouped DataFrames back together
    X_train = pd.concat(grouped_dfs)
    X_train.sort_index(inplace=True)
    print(X_train["group"].head())
```

```
X_train.drop(columns=to_drop, inplace=True)
X_test.drop(columns=to_drop, inplace=True)

X_train.to_csv('X_train_raw.csv', index=True)
X_test.to_csv('X_test_raw.csv', index=True)
```

```
absolute_humidity_2m:gm3              7.825
air_density_2m:kgm3                   1.245
ceiling_height_agl:m            2085.774902
clear_sky_energy_1h:J           1685498.875
clear_sky_rad:W                  452.100006
cloud_base_agl:m                2085.774902
dew_or_rime:idx                         0.0
dew_point_2m:K                   280.549988
diffuse_rad:W                    140.800003
diffuse_rad_1h:J                 538581.625
direct_rad:W                     102.599998
direct_rad_1h:J                 439453.8125
effective_cloud_cover:p           71.849998
elevation:m                             6.0
fresh_snow_12h:cm                       0.0
fresh_snow_1h:cm                        0.0
fresh_snow_24h:cm                       0.0
fresh_snow_3h:cm                        0.0
fresh_snow_6h:cm                        0.0
is_day:idx                              1.0
is_in_shadow:idx                        0.0
msl_pressure:hPa                1026.349976
precip_5min:mm                          0.0
precip_type_5min:idx                    0.0
pressure_100m:hPa               1013.325012
pressure_50m:hPa                1019.450012
prob_rime:p                             0.0
rain_water:kgm2                         0.0
relative_humidity_1000hPa:p       77.099998
sfc_pressure:hPa                1025.550049
snow_density:kgm3                       NaN
snow_depth:cm                           0.0
snow_drift:idx                          0.0
snow_melt_10min:mm                      0.0
snow_water:kgm2                         0.0
sun_azimuth:d                     93.415253
```

```
sun_elevation:d                     27.633499
super_cooled_liquid_water:kgm2          0.025
t_1000hPa:K                           282.625
total_cloud_cover:p                 71.849998
visibility:m                        44177.875
wind_speed_10m:ms                       2.675
wind_speed_u_10m:ms                      -2.3
wind_speed_v_10m:ms                      -1.4
wind_speed_w_1000hPa:ms                   0.0
is_estimated                                0
y                                     2991.12
location                                    A
Name: 2019-06-11 06:00:00, dtype: object
                     absolute_humidity_2m:gm3  air_density_2m:kgm3  \
ds
2019-06-02 22:00:00                       7.7              1.22825

                     ceiling_height_agl:m  clear_sky_energy_1h:J  \
ds
2019-06-02 22:00:00           1728.949951                    0.0

                     clear_sky_rad:W  cloud_base_agl:m  dew_or_rime:idx  \
ds
2019-06-02 22:00:00              0.0       1728.949951              0.0

                     dew_point_2m:K  diffuse_rad:W  diffuse_rad_1h:J  …  \
ds                                                                     …
2019-06-02 22:00:00      280.299988            0.0               0.0  …

                     t_1000hPa:K  total_cloud_cover:p  visibility:m  \
ds
2019-06-02 22:00:00   286.225006                100.0  40386.476562

                     wind_speed_10m:ms  wind_speed_u_10m:ms  \
ds
2019-06-02 22:00:00                3.6               -3.575

                     wind_speed_v_10m:ms  wind_speed_w_1000hPa:ms  \
ds
2019-06-02 22:00:00                 -0.5                      0.0

                     is_estimated    y  location
ds
2019-06-02 22:00:00             0  0.0         A

[1 rows x 48 columns]
```

[10]:
```python
# Create a plot of X_train showing its "y" and color it based on the value of␣
↪the sample_weight column.
if "sample_weight" in X_train.columns:
    import matplotlib.pyplot as plt
    import seaborn as sns
    sns.scatterplot(data=X_train, x=X_train.index, y="y", hue="sample_weight",␣
↪palette="deep", size=3)
    plt.show()
```

[11]:
```python
def normalize_sample_weights_per_location(df):
    for loc in locations:
        loc_df = df[df["location"] == loc]
        loc_df["sample_weight"] = loc_df["sample_weight"] /␣
↪loc_df["sample_weight"].sum() * loc_df.shape[0]
        df[df["location"] == loc] = loc_df
    return df

import pandas as pd

def split_and_shuffle_data(input_data, num_bins, frac1):
    """
    Splits the input_data into num_bins and shuffles them, then divides the␣
↪bins into two datasets based on the given fraction for the first set.

    Args:
        input_data (pd.DataFrame): The data to be split and shuffled.
        num_bins (int): The number of bins to split the data into.
        frac1 (float): The fraction of each bin to go into the first output␣
↪dataset.

    Returns:
        pd.DataFrame, pd.DataFrame: The two output datasets.
    """
    # Validate the input fraction
    if frac1 < 0 or frac1 > 1:
        raise ValueError("frac1 must be between 0 and 1.")

    if frac1==1:
        return input_data, pd.DataFrame()

    # Calculate the fraction for the second output set
    frac2 = 1 - frac1

    # Calculate bin size
    bin_size = len(input_data) // num_bins

    # Initialize empty DataFrames for output
```

```
    output_data1 = pd.DataFrame()
    output_data2 = pd.DataFrame()

    for i in range(num_bins):
        # Shuffle the data in the current bin
        np.random.seed(i)
        current_bin = input_data.iloc[i * bin_size: (i + 1) * bin_size].
 ↪sample(frac=1)

        # Calculate the sizes for each output set
        size1 = int(len(current_bin) * frac1)

        # Split and append to output DataFrames
        output_data1 = pd.concat([output_data1, current_bin.iloc[:size1]])
        output_data2 = pd.concat([output_data2, current_bin.iloc[size1:]])

    # Shuffle and split the remaining data
    remaining_data = input_data.iloc[num_bins * bin_size:].sample(frac=1)
    remaining_size1 = int(len(remaining_data) * frac1)

    output_data1 = pd.concat([output_data1, remaining_data.iloc[:
 ↪remaining_size1]])
    output_data2 = pd.concat([output_data2, remaining_data.iloc[remaining_size1:
 ↪]])

    return output_data1, output_data2
```

[12]:

```
from autogluon.tabular import TabularDataset, TabularPredictor
data = TabularDataset('X_train_raw.csv')
# set group column of train_data be increasing from 0 to 7 based on time, the␣
 ↪first 1/8 of the data is group 0, the second 1/8 of the data is group 1, etc.
data['ds'] = pd.to_datetime(data['ds'])
data = data.sort_values(by='ds')

# # print size of the group for each location
# for loc in locations:
#     print(f"Location {loc}:")
#     print(train_data[train_data["location"] == loc].groupby('group').size())


# get end date of train data and subtract 3 months
#split_time = pd.to_datetime(train_data["ds"]).max() - pd.
 ↪Timedelta(hours=tune_and_test_length)
# 2022-10-28 22:00:00
split_time = pd.to_datetime("2022-10-28 22:00:00")
train_set = TabularDataset(data[data["ds"] < split_time])
test_set = TabularDataset(data[data["ds"] >= split_time])
```

```python
# shuffle test_set and only grab tune_and_test_length percent of it, rest goes␣
↪to train_set
test_set, new_train_set = split_and_shuffle_data(test_set, 40,␣
↪tune_and_test_length)


print("Length of train set before adding test set", len(train_set))
# add rest to train_set
train_set = pd.concat([train_set, new_train_set])
print("Length of train set after adding test set", len(train_set))
print("Length of test set", len(test_set))




if use_groups:
    test_set = test_set.drop(columns=['group'])


tuning_data = None
if use_tune_data:
    if use_test_data:
        # split test_set in half, use first half for tuning
        tuning_data, test_data = [], []
        for loc in locations:
            loc_test_set = test_set[test_set["location"] == loc]
            # randomly shuffle the loc_test_set
            loc_tuning_data, loc_test_data =␣
↪split_and_shuffle_data(loc_test_set, 40, 0.5)
            tuning_data.append(loc_tuning_data)
            test_data.append(loc_test_data)
        tuning_data = pd.concat(tuning_data)
        test_data = pd.concat(test_data)
        print("Shapes of tuning and test", tuning_data.shape[0], test_data.
↪shape[0], tuning_data.shape[0] + test_data.shape[0])

    else:
        tuning_data = test_set
        print("Shape of tuning", tuning_data.shape[0])

    # ensure sample weights for your tuning data sum to the number of rows in␣
↪the tuning data.
    if weight_evaluation:
        tuning_data = normalize_sample_weights_per_location(tuning_data)
```

```
else:
    if use_test_data:
        test_data = test_set
        print("Shape of test", test_data.shape[0])


train_data = train_set

# ensure sample weights for your training (or tuning) data sum to the number of␣
↪rows in the training (or tuning) data.
if weight_evaluation:
    train_data = normalize_sample_weights_per_location(train_data)
    if use_test_data:
        test_data = normalize_sample_weights_per_location(test_data)


train_data = TabularDataset(train_data)
if use_tune_data:
    tuning_data = TabularDataset(tuning_data)
if use_test_data:
    test_data = TabularDataset(test_data)
```

```
Length of train set before adding test set 78668
Length of train set after adding test set 84074
Length of test set 5365
Shapes of tuning and test 2641 2724 5365
```

3 Quick EDA

[13]:
```
if run_analysis:
    import autogluon.eda.auto as auto
    auto.dataset_overview(train_data=train_data, test_data=test_data,␣
↪label="y", sample=None)
```

[14]:
```
if run_analysis:
    auto.target_analysis(train_data=train_data, label="y", sample=None)
```

4 Modeling

[15]:
```
import os


# Get the last submission number
last_submission_number = int(max([int(filename.split('_')[1].split('.')[0]) for␣
↪filename in os.listdir('submissions') if "submission" in filename]))
print("Last submission number:", last_submission_number)
```

```
print("Now creating submission number:", last_submission_number + 1)

# Create the new filename
new_filename = f'submission_{last_submission_number + 1}'

hello = os.environ.get('HELLO')
if hello is not None:
    new_filename += f'_{hello}'

print("New filename:", new_filename)
```

```
Last submission number: 103
Now creating submission number: 104
New filename: submission_104
```

[16]:
```
predictors = [None, None, None]
```

[17]:
```
def fit_predictor_for_location(loc):
    print(f"Training model for location {loc}...")
    # sum of sample weights for this location, and number of rows, for both␣
↪train and tune data and test data
    if weight_evaluation:
        print("Train data sample weight sum:",␣
↪train_data[train_data["location"] == loc]["sample_weight"].sum())
        print("Train data number of rows:", train_data[train_data["location"]␣
↪== loc].shape[0])
        if use_tune_data:
            print("Tune data sample weight sum:",␣
↪tuning_data[tuning_data["location"] == loc]["sample_weight"].sum())
            print("Tune data number of rows:",␣
↪tuning_data[tuning_data["location"] == loc].shape[0])
        if use_test_data:
            print("Test data sample weight sum:",␣
↪test_data[test_data["location"] == loc]["sample_weight"].sum())
            print("Test data number of rows:", test_data[test_data["location"]␣
↪== loc].shape[0])
    predictor = TabularPredictor(
        label=label,
        eval_metric=metric,
        path=f"AutogluonModels/{new_filename}_{loc}",
        # sample_weight=sample_weight,
        # weight_evaluation=weight_evaluation,
        # groups="group" if use_groups else None,
    ).fit(
        train_data=train_data[train_data["location"] == loc].
↪drop(columns=["ds"]),
        time_limit=time_limit,
```

```
        # presets=presets,
        num_stack_levels=num_stack_levels,
        num_bag_folds=num_bag_folds if not use_groups else 2,# just put␣
↪somethin, will be overwritten anyways
        num_bag_sets=num_bag_sets,
        tuning_data=tuning_data[tuning_data["location"] == loc].
↪reset_index(drop=True).drop(columns=["ds"]) if use_tune_data else None,
        use_bag_holdout=use_bag_holdout,
        # holdout_frac=holdout_frac,
    )

    # evaluate on test data
    if use_test_data:
        # drop sample_weight column
        t = test_data[test_data["location"] == loc]#.
↪drop(columns=["sample_weight"])
        perf = predictor.evaluate(t)
        print("Evaluation on test data:")
        print(perf[predictor.eval_metric.name])

    return predictor

loc = "A"
predictors[0] = fit_predictor_for_location(loc)
```

```
Beginning AutoGluon training … Time limit = 300s
AutoGluon will save models to "AutogluonModels/submission_104_A/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   93.71 GB / 315.93 GB (29.7%)
Train Data Rows:    31713
Train Data Columns: 32
Tuning Data Rows:    1054
Tuning Data Columns: 32
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (5733.42, 0.0, 654.3279,
1182.46326)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
```

```
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    128742.38 MB
        Train Data (Original)  Memory Usage: 10.03 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
```

```
Training model for location A…
```

```
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        30 features in original data used to generate 30 features in processed
data.
        Train Data (Processed) Memory Usage: 7.63 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.15s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
```

```
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 299.85s of the
299.85s of remaining time.
        -132.8646        = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.37s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 299.35s of the
299.35s of remaining time.
        -130.7925        = Validation score   (-mean_absolute_error)
        0.03s    = Training   runtime
        0.37s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 298.89s of the
298.89s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -86.6594         = Validation score   (-mean_absolute_error)
        28.43s   = Training   runtime
        12.95s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 261.29s of the
261.29s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -94.1378         = Validation score   (-mean_absolute_error)
        22.79s   = Training   runtime
        5.47s    = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 234.8s of the
234.79s of remaining time.
        -104.1952        = Validation score   (-mean_absolute_error)
```

```
        6.99s    = Training   runtime
        1.1s     = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 225.5s of the 225.5s
of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -100.0739        = Validation score   (-mean_absolute_error)
        180.55s  = Training   runtime
        0.17s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 43.77s of the
43.77s of remaining time.
        -106.4086        = Validation score   (-mean_absolute_error)
        1.65s    = Training   runtime
        1.08s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 39.64s of the
39.64s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -102.6302        = Validation score   (-mean_absolute_error)
        33.2s    = Training   runtime
        0.49s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 4.06s of the 4.06s of
remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -100.8576        = Validation score   (-mean_absolute_error)
        3.31s    = Training   runtime
        0.25s    = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 299.85s of the
-1.21s of remaining time.
        -86.0935         = Validation score   (-mean_absolute_error)
        0.36s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 301.6s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_104_A/")
Evaluation: mean_absolute_error on test data: -99.45476478975004
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -99.45476478975004,
    "root_mean_squared_error": -252.02443014922616,
    "mean_squared_error": -63516.313392042175,
    "r2": 0.9145463919194448,
    "pearsonr": 0.9563941553411301,
```

```
    "median_absolute_error": -5.23750114440918
}
```

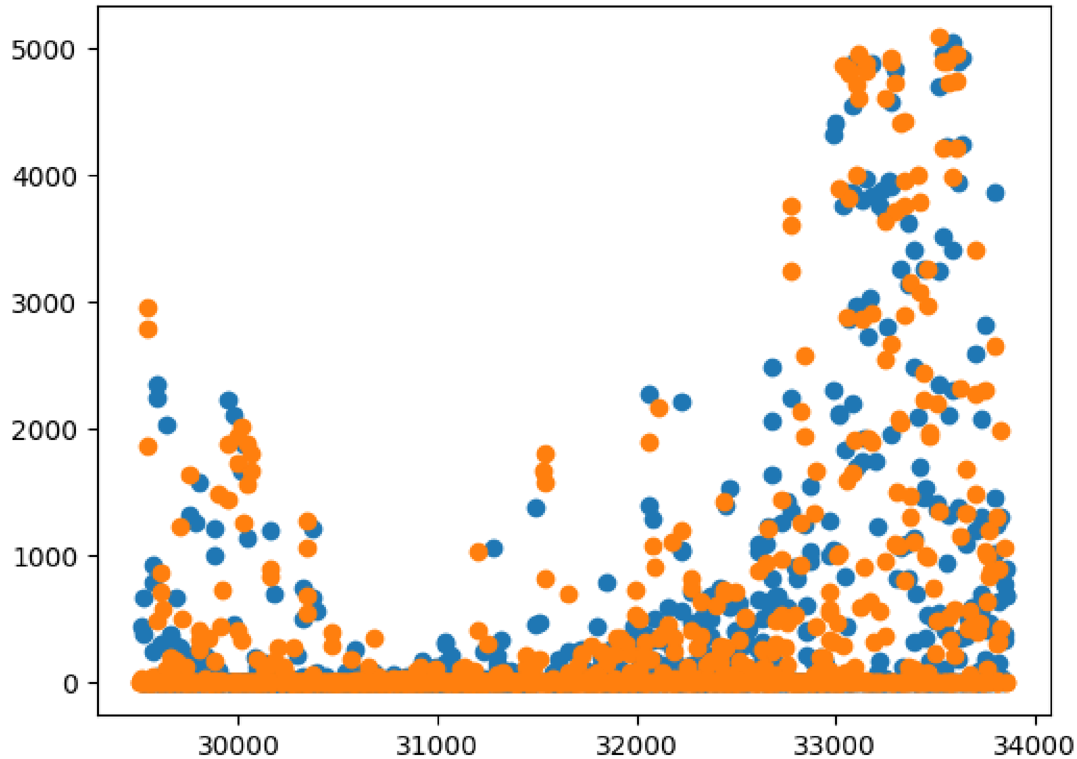
```
Evaluation on test data:
-99.45476478975004
```

[18]:

```python
import matplotlib.pyplot as plt

leaderboards = [None, None, None]
def leaderboard_for_location(i, loc):
    if use_test_data:
        lb = predictors[i].leaderboard(test_data[test_data["location"] == loc])
        lb["location"] = loc
        plt.scatter(test_data[test_data["location"] == loc]["y"].index,␣
 ↪test_data[test_data["location"] == loc]["y"])
        if use_tune_data:
            plt.scatter(tuning_data[tuning_data["location"] == loc]["y"].index,␣
 ↪tuning_data[tuning_data["location"] == loc]["y"])
        plt.show()

        return lb
    else:
        return pd.DataFrame()

leaderboards[0] = leaderboard_for_location(0, loc)
```

```
                     model  score_test   score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0      WeightedEnsemble_L2  -99.454765  -86.093515        1.959402      18.912609
84.779248                 0.003194                0.000654           0.359085
2        True          10
1        LightGBMXT_BAG_L1 -100.236089  -86.659392        1.088782      12.951687
28.434958                 1.088782               12.951687          28.434958
1        True           3
2          LightGBM_BAG_L1 -103.433549  -94.137774        0.606663       5.474789
22.788302                 0.606663                5.474789          22.788302
1        True           4
3           XGBoost_BAG_L1 -107.719150 -100.857591        0.115978       0.246350
3.314704                 0.115978                0.246350           3.314704
1        True           9
4          CatBoost_BAG_L1 -110.162882 -100.073897        0.067334       0.172364
180.547148                 0.067334                0.172364         180.547148
1        True           6
5   RandomForestMSE_BAG_L1 -110.384757 -104.195205        0.623186       1.097277
6.987688                 0.623186                1.097277           6.987688
1        True           5
6   NeuralNetFastAI_BAG_L1 -112.041584 -102.630172        0.260763       0.485479
```

```
33.196903                0.260763                0.485479          33.196903
1       True           8
7    ExtraTreesMSE_BAG_L1 -115.077057 -106.408572         0.613879        1.082442
1.645114                0.613879                1.082442          1.645114
1       True           7
8   KNeighborsDist_BAG_L1 -136.179430 -130.792507         0.019972        0.373317
0.029093                0.019972                0.373317          0.029093
1       True           2
9   KNeighborsUnif_BAG_L1 -136.234669 -132.864617         0.164141        0.374408
0.030098                0.164141                0.374408          0.030098
1       True           1
```



[19]:
```python
loc = "B"
predictors[1] = fit_predictor_for_location(loc)
leaderboards[1] = leaderboard_for_location(1, loc)
```

```
Beginning AutoGluon training … Time limit = 300s
AutoGluon will save models to "AutogluonModels/submission_104_B/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
```

```
Disk Space Avail:   92.44 GB / 315.93 GB (29.3%)
Train Data Rows:    27787
Train Data Columns: 32
Tuning Data Rows:    894
Tuning Data Columns: 32
Label Column: y
Preprocessing data …
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and many unique label-values observed).
        Label info (max, min, mean, stddev): (1152.3, -0.0, 96.80802, 205.20811)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    126805.64 MB
        Train Data (Original)  Memory Usage: 8.78 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
```

Training model for location B…

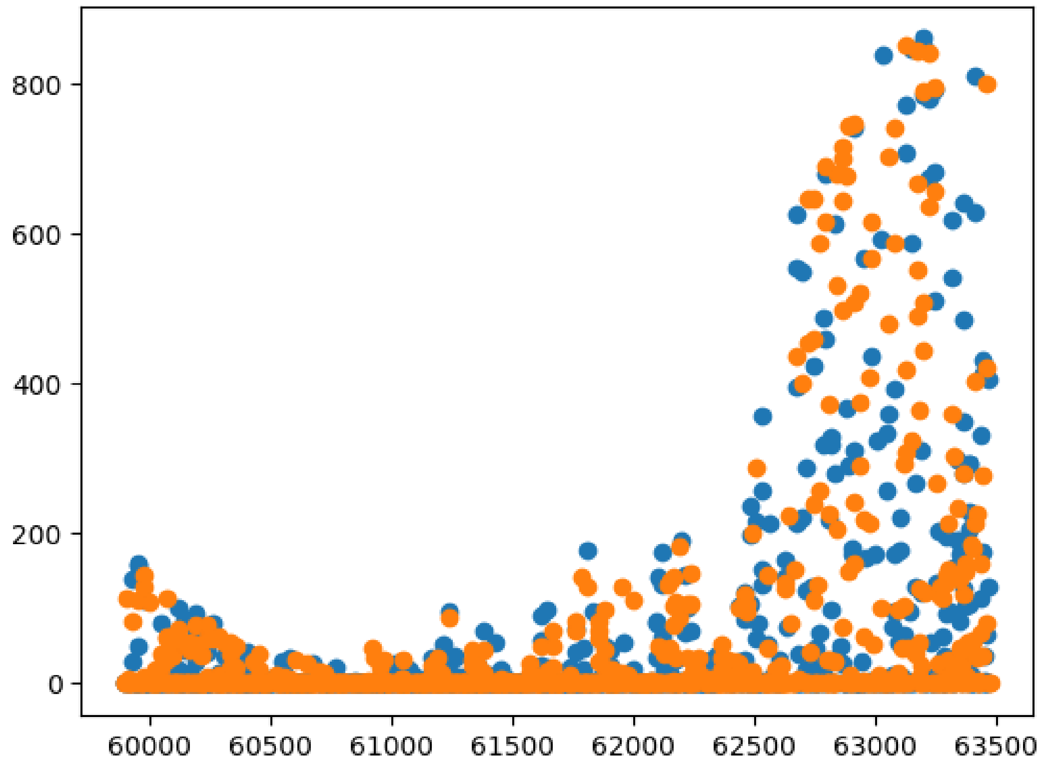
```
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
```

```
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        30 features in original data used to generate 30 features in processed
data.
        Train Data (Processed) Memory Usage: 6.68 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.14s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 299.86s of the
299.85s of remaining time.
        -21.7073         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        0.32s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 299.45s of the
299.45s of remaining time.
        -21.7889         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        0.38s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 298.99s of the
298.98s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
```

```
ParallelLocalFoldFittingStrategy
        -13.8447         = Validation score   (-mean_absolute_error)
        27.51s   = Training   runtime
        16.43s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 266.85s of the
266.84s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.6751         = Validation score   (-mean_absolute_error)
        26.25s   = Training   runtime
        5.0s     = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 236.96s of
the 236.95s of remaining time.
        -15.7746         = Validation score   (-mean_absolute_error)
        6.03s    = Training   runtime
        0.89s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 229.12s of the
229.12s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.2554         = Validation score   (-mean_absolute_error)
        184.05s  = Training   runtime
        0.09s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 43.82s of the
43.81s of remaining time.
        -15.801  = Validation score   (-mean_absolute_error)
        1.29s    = Training   runtime
        0.86s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 40.73s of the
40.73s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -13.7865         = Validation score   (-mean_absolute_error)
        33.64s   = Training   runtime
        0.5s     = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 5.64s of the 5.64s of
remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.8129         = Validation score   (-mean_absolute_error)
        5.21s    = Training   runtime
        0.36s    = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 299.86s of the
-1.21s of remaining time.
        -12.9923         = Validation score   (-mean_absolute_error)
        0.36s    = Training   runtime
        0.0s     = Validation runtime
```

```
AutoGluon training complete, total runtime = 301.59s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_104_B/")
Evaluation: mean_absolute_error on test data: -10.818144448352175
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
Evaluations on test data:
{
    "mean_absolute_error": -10.818144448352175,
    "root_mean_squared_error": -28.8941908467152,
    "mean_squared_error": -834.8742646864006,
    "r2": 0.9537820309086132,
    "pearsonr": 0.976658166965361,
    "median_absolute_error": -1.4715003967285156
}

Evaluation on test data:
-10.818144448352175
                    model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0     WeightedEnsemble_L2  -10.818144 -12.992342        1.897629      17.908468
251.580466                0.003573                0.000679           0.357679
2       True         10
1       LightGBMXT_BAG_L1  -11.368320 -13.844705        1.142890      16.427684
27.506043               1.142890               16.427684          27.506043
1       True          3
2         LightGBM_BAG_L1  -11.707436 -14.675131        0.769769       4.996075
26.254703               0.769769                4.996075          26.254703
1       True          4
3          XGBoost_BAG_L1  -11.731573 -14.812939        0.139304       0.357983
5.205399               0.139304                0.357983           5.205399
1       True          9
4  NeuralNetFastAI_BAG_L1  -12.038627 -13.786499        0.241209       0.500549
33.640361               0.241209                0.500549          33.640361
1       True          8
5         CatBoost_BAG_L1  -12.389705 -14.255426        0.074729       0.091246
184.047900                0.074729                0.091246         184.047900
1       True          6
6  RandomForestMSE_BAG_L1  -12.941215 -15.774592        0.435228       0.888310
6.028483               0.435228                0.888310           6.028483
1       True          5
7    ExtraTreesMSE_BAG_L1  -13.174879 -15.801002        0.474827       0.864240
1.293498               0.474827                0.864240           1.293498
1       True          7
8   KNeighborsUnif_BAG_L1  -18.767645 -21.707274        0.022875       0.321353
0.024787               0.022875                0.321353           0.024787
```

```
1       True           1
9   KNeighborsDist_BAG_L1  -18.893222 -21.788906        0.019319        0.377714
0.024272                 0.019319                0.377714            0.024272
1       True           2
```



[20]:
```
loc = "C"
predictors[2] = fit_predictor_for_location(loc)
leaderboards[2] = leaderboard_for_location(2, loc)
```

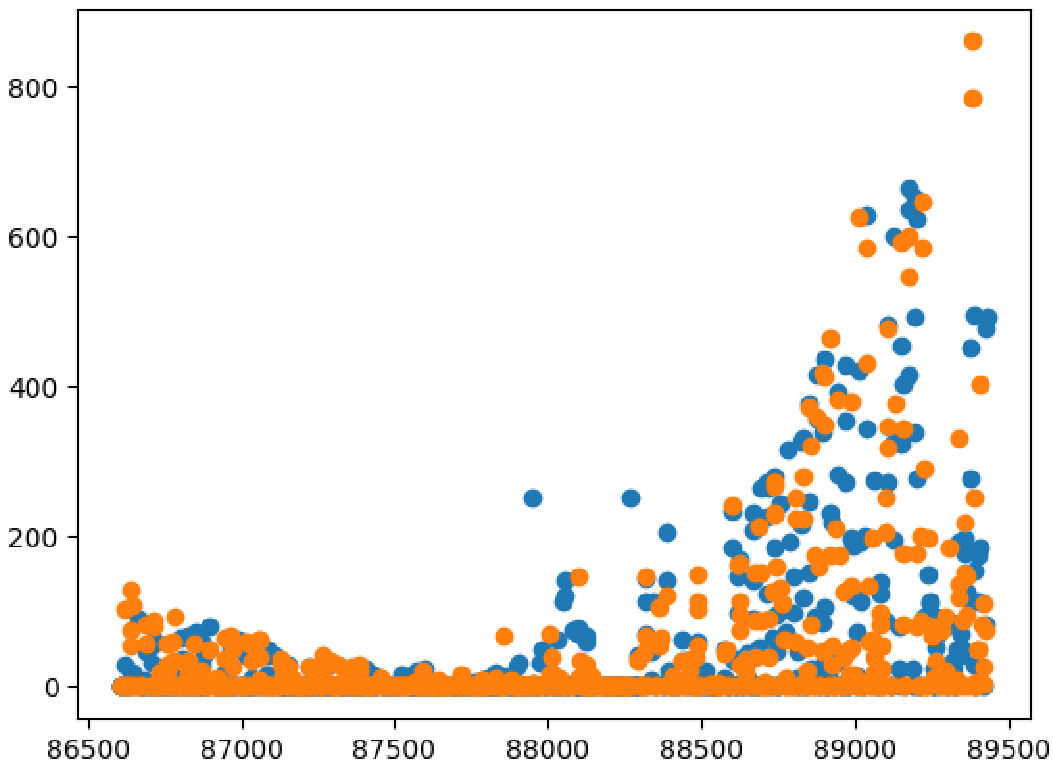
```
Beginning AutoGluon training … Time limit = 300s
AutoGluon will save models to "AutogluonModels/submission_104_C/"
AutoGluon Version:  0.8.2
Python Version:     3.10.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Debian 5.10.197-1 (2023-09-29)
Disk Space Avail:   91.37 GB / 315.93 GB (28.9%)
Train Data Rows:    24574
Train Data Columns: 32
Tuning Data Rows:    693
Tuning Data Columns: 32
Label Column: y
Preprocessing data …
```

```
AutoGluon infers your prediction problem is: 'regression' (because dtype of
label-column == float and label-values can't be converted to int).
        Label info (max, min, mean, stddev): (999.6, 0.0, 79.99842, 168.73961)
        If 'regression' is not the correct problem_type, please manually specify
the problem_type parameter during predictor init (You may specify problem_type
as one of: ['binary', 'multiclass', 'regression'])
Using Feature Generators to preprocess the data …
Fitting AutoMLPipelineFeatureGenerator…
        Available Memory:                    126604.95 MB
        Train Data (Original)  Memory Usage: 7.73 MB (0.0% of available memory)
        Inferring data type of each feature based on column values. Set
feature_metadata_in to manually specify special dtypes of the features.
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator…
                        Note: Converting 1 features to boolean dtype as they
only contain 2 unique values.
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator…
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator…
        Stage 4 Generators:
                Fitting DropUniqueFeatureGenerator…
        Stage 5 Generators:
                Fitting DropDuplicatesFeatureGenerator…
        Useless Original Features (Count: 2): ['elevation:m', 'location']
                These features carry no predictive signal and should be manually
investigated.
                This is typically a feature which has the same value for all
rows.
                These features do not need to be present at inference time.
        Types of features in original data (raw dtype, special dtypes):
                ('float', []) : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', [])   :  1 | ['is_estimated']
        Types of features in processed data (raw dtype, special dtypes):
                ('float', [])     : 29 | ['ceiling_height_agl:m',
'clear_sky_energy_1h:J', 'clear_sky_rad:W', 'cloud_base_agl:m', 'diffuse_rad:W',
…]
                ('int', ['bool']) :  1 | ['is_estimated']
        0.1s = Fit runtime
        30 features in original data used to generate 30 features in processed
data.
        Train Data (Processed) Memory Usage: 5.89 MB (0.0% of available memory)
Data preprocessing and feature engineering runtime = 0.14s …
AutoGluon will gauge predictive performance using evaluation metric:
'mean_absolute_error'
        This metric's sign has been flipped to adhere to being higher_is_better.
```

```
The metric score can be multiplied by -1 to get the metric value.
        To change this, specify the eval_metric parameter of Predictor()
use_bag_holdout=True, will use tuning_data as holdout (will not be used for
early stopping).
User-specified model hyperparameters to be fit:
{
        'NN_TORCH': {},
        'GBM': [{'extra_trees': True, 'ag_args': {'name_suffix': 'XT'}}, {},
'GBMLarge'],
        'CAT': {},
        'XGB': {},
        'FASTAI': {},
        'RF': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'XT': [{'criterion': 'gini', 'ag_args': {'name_suffix': 'Gini',
'problem_types': ['binary', 'multiclass']}}, {'criterion': 'entropy', 'ag_args':
{'name_suffix': 'Entr', 'problem_types': ['binary', 'multiclass']}},
{'criterion': 'squared_error', 'ag_args': {'name_suffix': 'MSE',
'problem_types': ['regression', 'quantile']}}],
        'KNN': [{'weights': 'uniform', 'ag_args': {'name_suffix': 'Unif'}},
{'weights': 'distance', 'ag_args': {'name_suffix': 'Dist'}}],
}
Fitting 11 L1 models …
Fitting model: KNeighborsUnif_BAG_L1 … Training model for up to 299.86s of the
299.86s of remaining time.

Training model for location C…

        -25.5888         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        0.27s    = Validation runtime
Fitting model: KNeighborsDist_BAG_L1 … Training model for up to 299.5s of the
299.5s of remaining time.
        -25.6861         = Validation score   (-mean_absolute_error)
        0.02s    = Training   runtime
        0.25s    = Validation runtime
Fitting model: LightGBMXT_BAG_L1 … Training model for up to 299.16s of the
299.16s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -12.5617         = Validation score   (-mean_absolute_error)
        25.56s   = Training   runtime
        11.19s   = Validation runtime
Fitting model: LightGBM_BAG_L1 … Training model for up to 269.2s of the
269.19s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
```

```
ParallelLocalFoldFittingStrategy
        -13.4211         = Validation score   (-mean_absolute_error)
        25.38s   = Training   runtime
        6.0s     = Validation runtime
Fitting model: RandomForestMSE_BAG_L1 … Training model for up to 240.34s of
the 240.34s of remaining time.
        -18.1676         = Validation score   (-mean_absolute_error)
        4.39s    = Training   runtime
        0.75s    = Validation runtime
Fitting model: CatBoost_BAG_L1 … Training model for up to 234.53s of the
234.52s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.1573         = Validation score   (-mean_absolute_error)
        188.31s  = Training   runtime
        0.08s    = Validation runtime
Fitting model: ExtraTreesMSE_BAG_L1 … Training model for up to 44.96s of the
44.96s of remaining time.
        -17.0326         = Validation score   (-mean_absolute_error)
        0.98s    = Training   runtime
        0.78s    = Validation runtime
Fitting model: NeuralNetFastAI_BAG_L1 … Training model for up to 42.45s of the
42.45s of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.3042         = Validation score   (-mean_absolute_error)
        30.88s   = Training   runtime
        0.41s    = Validation runtime
Fitting model: XGBoost_BAG_L1 … Training model for up to 10.07s of the 10.07s
of remaining time.
        Fitting 8 child models (S1F1 - S1F8) | Fitting with
ParallelLocalFoldFittingStrategy
        -14.9796         = Validation score   (-mean_absolute_error)
        8.58s    = Training   runtime
        0.38s    = Validation runtime
Completed 1/20 k-fold bagging repeats …
Fitting model: WeightedEnsemble_L2 … Training model for up to 299.86s of the
-0.23s of remaining time.
        -12.3814         = Validation score   (-mean_absolute_error)
        0.34s    = Training   runtime
        0.0s     = Validation runtime
AutoGluon training complete, total runtime = 300.6s … Best model:
"WeightedEnsemble_L2"
TabularPredictor saved. To load, use: predictor =
TabularPredictor.load("AutogluonModels/submission_104_C/")
Evaluation: mean_absolute_error on test data: -11.783568601074712
        Note: Scores are always higher_is_better. This metric score can be
multiplied by -1 to get the metric value.
```

```
Evaluations on test data:
{
    "mean_absolute_error": -11.783568601074712,
    "root_mean_squared_error": -29.077424156048103,
    "mean_squared_error": -845.4965955507297,
    "r2": 0.9256248933070587,
    "pearsonr": 0.9627813000374813,
    "median_absolute_error": -1.197532832622528
}

Evaluation on test data:
-11.783568601074712
                    model  score_test  score_val  pred_time_test  pred_time_val
fit_time  pred_time_test_marginal  pred_time_val_marginal  fit_time_marginal
stack_level  can_infer  fit_order
0       LightGBMXT_BAG_L1  -11.767180 -12.561670        0.993090      11.186538
25.564893                 0.993090               11.186538          25.564893
1       True          3
1     WeightedEnsemble_L2  -11.783569 -12.381402        1.200459      11.597554
56.785779                 0.002607                0.000648           0.343755
2       True         10
2         LightGBM_BAG_L1  -13.015381 -13.421080        0.765038       5.999748
25.375116                 0.765038                5.999748          25.375116
1       True          4
3         CatBoost_BAG_L1  -13.139805 -14.157315        0.074831       0.084700
188.306934                 0.074831                0.084700         188.306934
1       True          6
4          XGBoost_BAG_L1  -14.157795 -14.979604        0.148847       0.382866
8.583807                 0.148847               0.382866           8.583807
1       True          9
5  NeuralNetFastAI_BAG_L1  -15.217472 -14.304202        0.204762       0.410369
30.877131                 0.204762                0.410369          30.877131
1       True          8
6    ExtraTreesMSE_BAG_L1  -16.373132 -17.032575        0.365627       0.782704
0.984232                 0.365627               0.782704           0.984232
1       True          7
7  RandomForestMSE_BAG_L1  -17.446433 -18.167550        0.337863       0.753439
4.389629                 0.337863               0.753439           4.389629
1       True          5
8   KNeighborsUnif_BAG_L1  -24.633067 -25.588795        0.012986       0.272817
0.024324                 0.012986               0.272817           0.024324
1       True          1
9   KNeighborsDist_BAG_L1  -24.751964 -25.686064        0.012697       0.253415
0.023448                 0.012697               0.253415           0.023448
1       True          2
```



[21]:
```
# save leaderboards to csv
pd.concat(leaderboards).to_csv(f"leaderboards/{new_filename}.csv")
```

5 Submit

[22]:
```
import pandas as pd
import matplotlib.pyplot as plt

future_test_data = TabularDataset('X_test_raw.csv')
future_test_data["ds"] = pd.to_datetime(future_test_data["ds"])
#test_data
```

```
Loaded data from: X_test_raw.csv | Columns = 33 / 33 | Rows = 4608 -> 4608
```

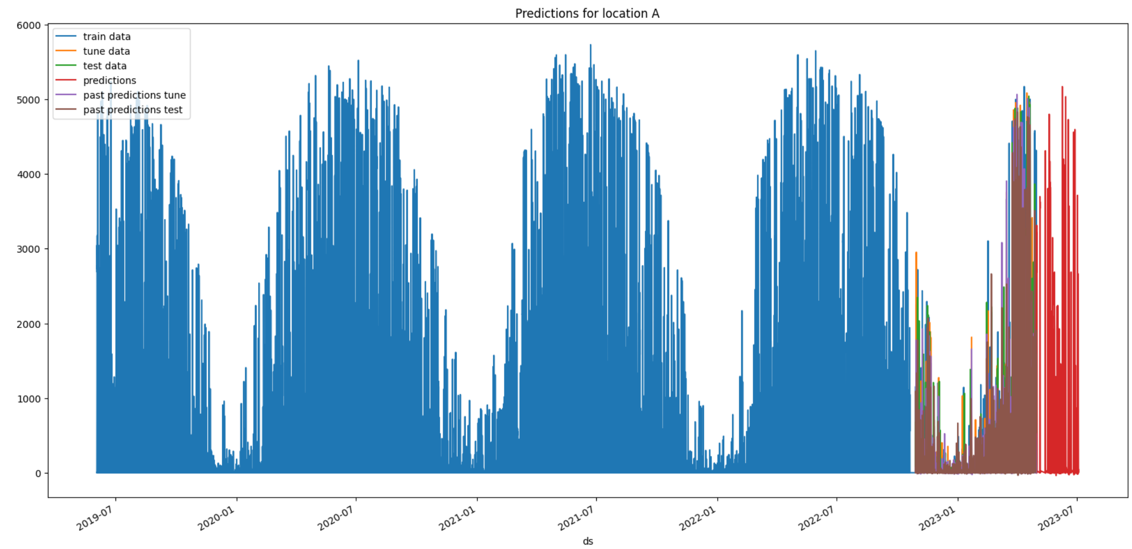
[23]:
```
test_ids = TabularDataset('test.csv')
test_ids["time"] = pd.to_datetime(test_ids["time"])
# merge test_data with test_ids
future_test_data_merged = pd.merge(future_test_data, test_ids, how="inner",␣
↪right_on=["time", "location"], left_on=["ds", "location"])

#test_data_merged
```

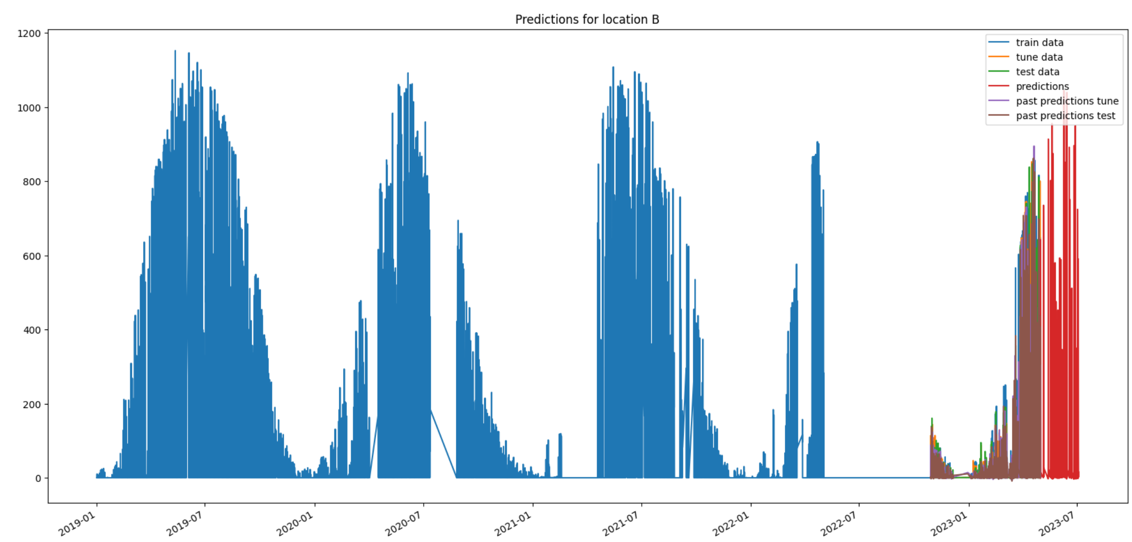
```
Loaded data from: test.csv | Columns = 4 / 4 | Rows = 2160 -> 2160
```

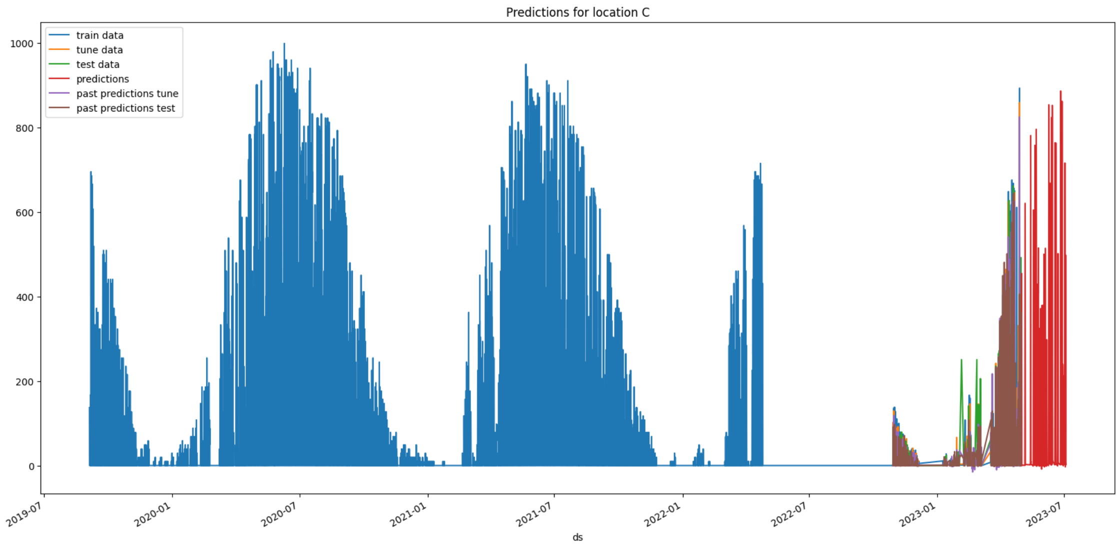
[24]:
```
# predict, grouped by location
predictions = []
location_map = {
    "A": 0,
    "B": 1,
    "C": 2
}
for loc, group in future_test_data.groupby('location'):
    i = location_map[loc]
    subset = future_test_data_merged[future_test_data_merged["location"] ==␣
↪loc].reset_index(drop=True)
    #print(subset)
    pred = predictors[i].predict(subset)
    subset["prediction"] = pred
    predictions.append(subset)

    # get past predictions
    #train_data.loc[train_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(train_data[train_data["location"] == loc])
    if use_tune_data:
        tuning_data.loc[tuning_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(tuning_data[tuning_data["location"] == loc])
    if use_test_data:
        test_data.loc[test_data["location"] == loc, "prediction"] = ␣
↪predictors[i].predict(test_data[test_data["location"] == loc])
```

[25]:
```
# plot predictions for location A, in addition to train data for A
for loc, idx in location_map.items():
    fig, ax = plt.subplots(figsize=(20, 10))
    # plot train data
    train_data[train_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
↪label="train data")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
↪label="tune data")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='y', ax=ax,␣
↪label="test data")

    # plot predictions
    predictions[idx].plot(x='ds', y='prediction', ax=ax, label="predictions")

    # plot past predictions
    #train_data_with_dates[train_data_with_dates["location"]==loc].plot(x='ds',␣
↪y='prediction', ax=ax, label="past predictions")
```

```
    #train_data[train_data["location"]==loc].plot(x='ds', y='prediction', ax=ax, label="past predictions train")
    if use_tune_data:
        tuning_data[tuning_data["location"]==loc].plot(x='ds', y='prediction', ax=ax, label="past predictions tune")
    if use_test_data:
        test_data[test_data["location"]==loc].plot(x='ds', y='prediction', ax=ax, label="past predictions test")


    # title
    ax.set_title(f"Predictions for location {loc}")
```







[26]:
```
temp_predictions = [prediction.copy() for prediction in predictions]
if clip_predictions:
    # clip predictions smaller than 0 to 0
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        pred.loc[pred["prediction"] < 0, "prediction"] = 0
        print("Smallest prediction after clipping:", pred["prediction"].min())

# Instead of clipping, shift all prediction values up by the largest negative␣
↪number.
# This way, the smallest prediction will be 0.
elif shift_predictions:
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        pred["prediction"] = pred["prediction"] - pred["prediction"].min()
        print("Smallest prediction after clipping:", pred["prediction"].min())

elif shift_predictions_by_average_of_negatives_then_clip:
    for pred in temp_predictions:
        # print smallest prediction
        print("Smallest prediction:", pred["prediction"].min())
        mean_negative = pred[pred["prediction"] < 0]["prediction"].mean()
        # if not nan
        if mean_negative == mean_negative:
```

```
            pred["prediction"] = pred["prediction"] - mean_negative

        pred.loc[pred["prediction"] < 0, "prediction"] = 0
        print("Smallest prediction after clipping:", pred["prediction"].min())



# concatenate predictions
submissions_df = pd.concat(temp_predictions)
submissions_df = submissions_df[["id", "prediction"]]
submissions_df
```

```
Smallest prediction: -38.522285
Smallest prediction after clipping: 0.0
Smallest prediction: -3.7432718
Smallest prediction after clipping: 0.0
Smallest prediction: -8.43025
Smallest prediction after clipping: 0.0
```

[26]:

```
       id  prediction
0       0    0.000000
1       1    0.000000
2       2    0.000000
3       3   50.458145
4       4  191.482712
..    …         …
715  2155   73.163185
716  2156   42.465805
717  2157    8.341773
718  2158    1.656495
719  2159    3.265739

[2160 rows x 2 columns]
```

[27]:

```
# Save the submission DataFrame to submissions folder, create new name based on
 ↪last submission, format is submission_<last_submission_number + 1>.csv

# Save the submission
print(f"Saving submission to submissions/{new_filename}.csv")
submissions_df.to_csv(os.path.join('submissions', f"{new_filename}.csv"),
 ↪index=False)
print("jall1a")
```

```
Saving submission to submissions/submission_104.csv
jall1a
```

[ ]:

```
# feature importance
```

```
# print starting calculating feature importance for location A with big text␣
↪font
print("\033[1m" + "Calculating feature importance for location A..." +␣
↪"\033[0m")
predictors[0].feature_importance(feature_stage="original",␣
↪data=test_data[test_data["location"] == "A"], time_limit=60*10)
print("\033[1m" + "Calculating feature importance for location B..." +␣
↪"\033[0m")
predictors[1].feature_importance(feature_stage="original",␣
↪data=test_data[test_data["location"] == "B"], time_limit=60*10)
print("\033[1m" + "Calculating feature importance for location C..." +␣
↪"\033[0m")
predictors[2].feature_importance(feature_stage="original",␣
↪data=test_data[test_data["location"] == "C"], time_limit=60*10)
```

```
These features in provided data are not utilized by the predictor and will be
ignored: ['ds', 'elevation:m', 'location', 'prediction']
Computing feature importance via permutation shuffling for 30 features using
1095 rows with 10 shuffle sets… Time limit: 600s…
```

Calculating feature importance for location A…

```
        608.58s = Expected runtime (60.86s per shuffle set)
```

[ ]:
```
# save this notebook to submissions folder
import subprocess
import os
subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
↪join('notebook_pdfs', f"{new_filename}_automatic_save.pdf"),␣
↪"autogluon_each_location.ipynb"])
#subprocess.run(["jupyter", "nbconvert", "--to", "pdf", "--output", os.path.
↪join('notebook_pdfs', f"{new_filename}.pdf"), "autogluon_each_location.
↪ipynb"])
```

[ ]:
```
# import subprocess

# def execute_git_command(directory, command):
#     """Execute a Git command in the specified directory."""
#     try:
#         result = subprocess.check_output(['git', '-C', directory] + command,␣
↪stderr=subprocess.STDOUT)
#         return result.decode('utf-8').strip(), True
#     except subprocess.CalledProcessError as e:
#         print(f"Git command failed with message: {e.output.decode('utf-8').
↪strip()}")
#         return e.output.decode('utf-8').strip(), False

# git_repo_path = "."
```

```
# execute_git_command(git_repo_path, ['config', 'user.email',␣
↪'henrikskog01@gmail.com'])
# execute_git_command(git_repo_path, ['config', 'user.name', hello if hello is␣
↪not None else 'Henrik eller Jørgen'])

# branch_name = new_filename

# # add datetime to branch name
# branch_name += f"_{pd.Timestamp.now().strftime('%Y-%m-%d_%H-%M-%S')}"

# commit_msg = "run result"

# execute_git_command(git_repo_path, ['checkout', '-b',branch_name])

# # Navigate to your repo and commit changes
# execute_git_command(git_repo_path, ['add', '.'])
# execute_git_command(git_repo_path, ['commit', '-m',commit_msg])

# # Push to remote
# output, success = execute_git_command(git_repo_path, ['push',␣
↪'origin',branch_name])

# # If the push fails, try setting an upstream branch and push again
# if not success and 'upstream' in output:
#     print("Attempting to set upstream and push again...")
#     execute_git_command(git_repo_path, ['push', '--set-upstream',␣
↪'origin',branch_name])
#     execute_git_command(git_repo_path, ['push', 'origin', 'henrik_branch'])

# execute_git_command(git_repo_path, ['checkout', 'main'])
```